120

AF/ 2122

OIPE JC60
MAY 2 4 2004
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Edmark et al.**	§	Group Art Unit: **2122**
	§	
Serial No.: **09/612,350**	§	Examiner: **Kendall, Chuck O.**
	§	
Filed: **July 6, 2000**	§	Attorney Docket No.: **AUS000057US1**
	§	

For: **Method and System for Tracing Profiling Information Using Per Thread Metric Variables with Reused Kernel Threads**

35525
PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

Certificate of Mailing Under 37 C.F.R. § 1.8(a)
I hereby certify this correspondence is being deposited with the United States Postal Service as First Class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on May 21, 2004.

By: Amelia C Turner
Amelia C. Turner

RECEIVED
MAY 2 7 2004
Technology Center 2100

TRANSMITTAL DOCUMENT

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:
ENCLOSED HEREWITH:

- Appellant's Brief (in triplicate) (37 C.F.R. 1.192); and
- Our return postcard.

A fee of $330.00 is required for filing an Appellant's Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0447. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0447. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0447.

Respectfully submitted,

[illegible]

Duke W. Yee
Registration No. 34,285
YEE & ASSOCIATES, P.C.
P.O. Box 802333
Dallas, Texas 75380
(972) 367-2001
ATTORNEY FOR APPLICANTS

OIPE JC60
MAY 24 2004
PATENT & TRADEMARK OFFICE

Docket No. AUS000057US1 *PATENT*

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Edmark et al.**	§	
	§	Group Art Unit: **2122**
Serial No. **09/612,350**	§	
	§	Examiner: **Kendall, Chuck O.**
Filed: **July 6, 2000**	§	
	§	
For: **Method and System for Tracing Profiling Information Using Per Thread Metric Variables with Reused Kernel Threads**	§ §	

RECEIVED
MAY 27 2004
Technology Center 2100

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

ATTENTION: Board of Patent Appeals and Interferences

<u>Certificate of Mailing Under 37 C.F.R. § 1.8(a)</u>
I hereby certify this correspondence is being deposited with the United States Postal Service as First Class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on May 21, 2004.

By: Amelia C Turner
Amelia C. Turner

APPELLANT'S BRIEF (37 C.F.R. 1.192)

This brief is in furtherance of the Notice of Appeal, filed in this case on March 22, 2004.

The fees required under § 1.17(c), and any required petition for extension of time for filing this brief and fees therefore, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

This brief is transmitted in triplicate. (37 C.F.R. 1.192(a))

05/26/2004 DEMMANU1 00000095 090447 09612350
01 FC:1402 330.00 DA

REAL PARTIES IN INTEREST

The real party in interest in this appeal is the following party: International Business Machines, Inc.

RELATED APPEALS AND INTERFERENCES

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-12, 14-21, and 23-34.

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims canceled: 13 and 22
2. Claims withdrawn from consideration but not canceled: NONE
3. Claims pending: 1-12, 14-21, and 23-34
4. Claims allowed: NONE
5. Claims rejected: 1-12, 14-21, and 23-34

C. CLAIMS ON APPEAL

The claims on appeal are: 1-12, 14-21, and 23-34.

STATUS OF AMENDMENTS

No amendments to the claims have been made after the Final Office Action.

SUMMARY OF INVENTION

The present invention is directed to a method and system for tracing profiling information using per metric variables with reused kernel threads. The invention addresses the problem when an operating system kernel reuses a kernel thread for a current thread and thus, it cannot be determined for certain whether the value of the kernel thread's metrics should be attributed to the current thread in its entirety. In one exemplary embodiment of the present invention, the invention operates by determining, while profiling, whether a new node has been created for a current thread that has a kernel thread ID that is already present in a hash table. If so, then the kernel thread ID is being reused. It is assumed that if an entry exists for the kernel thread ID in the hash table, then the kernel thread identity was used by another thread. On that assumption, the change in a metric may be attributed to a previous thread and applied to a previous thread's node.

ISSUES

The only issue is whether all of claims 1-12, 14-21 and 23-34 are anticipated under 35 U.S.C. § 102(e) by Berry et al. (U.S. Patent No. 6,539,339).

GROUPING OF CLAIMS

The claims do not stand or fall together. The claims stand or fall in accordance with the following grouping of claims with reasons for these groupings being provided in the following arguments:

Group I - claims 1, 10 and 11;
Group II – claim 2;
Group III – claim 3;

Group IV – claim 4;

Group V – claim 5;

Group VI – claim 6;

Group VII – claim 7;

Group VIII – claim 8;

Group IX - claim 9;

Group X - claims 12, 15, 17, 18, 21, 24 and 26-28;

Group XI - claims 14 and 23;

Group XII - claims 16 and 25;

Group XIII - claim 19;

Group XIV - claim 20;

Group XV - claims 29, 31 and 33; and

Group XVI – claims 30, 32 and 34.

ARGUMENT

The Final Office Action rejects claims 1-28 under 35 U.S.C. § 102(e) as being allegedly anticipated by Berry et al. (U.S. Patent No. 6,539,339). This rejection is respectfully traversed.

A. Claims 1-11

As per independent claim 1, the Office Action states:

> Regarding claim1 Berry anticipates, monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:
> receiving an event indication (3:35-40);
> ascertaining kernel thread level profile information (3:40-47);
> identifying a kernel thread, wherein the kernel thread level profile information is attributed to the identified kernel thread (3:40-47);
> determining whether the identified kernel thread has been reused (17:3-5, see recursion depth for reuse); and
> updating profile information with the kernel thread level profile information based whether the identified kernel thread has been reused (fig. 22, 2216).

Claim 1 reads as follows:

> 1. A method for monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:
> receiving an event indication;
> ascertaining kernel thread level profile information;
> identifying a kernel thread, wherein the kernel thread level profile information is attributed to the identified kernel thread;
> determining whether the identified kernel thread has been reused; and
> updating profile information with the kernel thread level profile information based whether the identified kernel thread has been reused. (emphasis added)

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.,* 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). Appellants respectfully submit that Berry does not identically show each and every feature of the present claims arranged as they are in the claims. Specifically, Berry does not teach determining whether the identified kernel thread has been reused or updating profile information with the kernel thread level profile information based on whether the identified kernel thread has been reused, as recited in claim 1.

Berry is directed to a method and system for maintaining a thread-relative metric for trace data using device driver support. With the method and system of Berry, a profiling process may detect a current event and, in response to the event, may request an elapsed metric since a preceding event. The profiling process then receives a thread-relative elapsed metric and may output a trace record for the current event in which is stored a metric equal to the received thread-relative elapsed metric. In response to a notification of an occurrence of the event, a device driver computes the thread relative elapsed metric by determining a current thread, retrieving a stored metric for the preceding event of the current thread, obtaining a current metric, and computing the thread-relative elapsed metric as a difference between the current metric and the stored metric.

Berry does not teach determining whether the identified kernel thread has been reused, as recited in claim 1. It is a basic tenet that the specification may be used as a dictionary for the terms used in the claims and that the claims must be read in light of the specification. As noted in the present specification, the term "reused" refers to a kernel thread being referenced by at least two different application threads (see page 53, line 26 to page 54, line 11 and page 61, line 29 to page 62, line 1 of the present specification, for example). Thus, the present invention as recited in claim 1, when read in light of the present specification, includes the step of determining whether a kernel thread has been referenced by more than one application thread. The Berry reference does not teach such a feature because Berry is not directed to solving the problem associated with the accumulation of metric values for reused kernel threads. To the contrary, the Berry reference is directed to solving the problem of an application thread switch occurring during an attempt to retrieve the time of an event (see Berry, column 3, lines 23-32).

The Office Action alleges that Berry teaches determining whether an identified kernel thread has been reused at column 17, lines 3-5 since this section of Berry allegedly teaches a recursion depth value and the Final Office Action equates this recursion depth value as being reuse of a kernel thread. Appellants respectfully disagree.

Column 17, lines 3-5 read as follows:

> The cumulative time in the example shown in FIG. 11B is four units of time. Finally, the recursion depth of call stack CAB is one, as none of the three routines present in the call stack have been recursively entered.

This section of Berry has nothing to do with whether a kernel thread is reused by application threads. To the contrary, all that this section of Berry is teaching is that the call stack may have an associated value that identifies how many times a routine is recursively entered. Recursion is the ability of a routine to call itself. Thus, the last value for each node in the tree data structure shown in Figure 11B of Berry represents how many times a routine calls itself. There is no correlation between this teaching in Berry and the feature of determining whether a kernel thread is reused by application threads in the present claims. The number of times a routine is recursively entered in itself has no bearing on whether a kernel thread is reused or not. Berry simply does not teach determining whether a kernel thread is reused or not. Since Berry does not teach determining whether a kernel thread is reused, Berry cannot teach updating profile information with the kernel thread level profile information based on whether the identified kernel thread has been reused.

These arguments were presented in the Response filed September 3, 2003. In response to these arguments, the Examiner, in the Final Office Action, states:

> (1) In claim 1 Applicant has argued that, prior art doesn't teach or disclose determining whether the identified kernel thread has been reused.
>
> Response (1) As set forth in claim 1, above Berry shows in Col. 3, lines 23-25 and lines 42-45, notification of event occurrences. Examiner believes this is the same as Applicant's limitation of whether the thread has been reused.
>
> (2) In claim 1 Applicant also argues that, prior art doesn't teach or disclose updating profile information with the kernel thread level profile information"
>
> Response (2) As set forth above in claim 1, Berry shows in Col. 24, lines 15-30, adding, updating and maintaining information regarding threads and also time stamp for the thread which it places in a trace record.
>
> (Final Office Action, page 8, "Response to Arguments")

Thus, in the Examiner's response to Appellants' arguments, the Examiner changes the sections of the Berry reference that allegedly teach the features of the present claims from column 17, lines 3-5 discussed above, to column 3, lines 23-25 and 42-45. These cited sections of Berry read as follows:

> An important datum to be recorded about the occurrence of an event is its time of occurrence. However, during the attempt to retrieve the time of an event, a thread switch may occur, and the recorded time may not accurately reflect the actual time that the event occurred.
> (Column 3, lines 23-27)
>
> In response to a notification of an occurrence of the current event, a device driver computes the thread-relative elapsed metric by determining a current thread; retrieving a stored metric for the preceding event of the current thread; obtaining a current metric; and computing the thread-relative elapsed metric as a difference between the current metric and the stored metric.
> (Column 3, lines 42-49)

Neither of these sections teaches anything having to do with reuse of a kernel thread. To the contrary, these sections are directed to determining the timing metrics around an event when there is an application thread switch, i.e. a change in execution from one application thread to a different application thread. These sections of Berry only serve to support Appellants' position that Berry actually has nothing to do with kernel thread reuse and is only concerned with determining timing metrics associated with switches between different application threads during the occurrence of an event.

An application thread switch is not the same as kernel thread reuse. As stated above, application thread switches are a change in processor execution from one application thread to a different application thread. Kernel thread reuse is when a kernel thread has been referenced by more than one application thread. There is no ability in Berry to determine whether a kernel thread has been reused. Furthermore, Berry has no concern as to whether a kernel thread has been reused. Other than generally dealing with metrics of threads, Berry is completely irrelevant to the presently claimed invention as set forth in claim 1.

With regard to the feature of updating profile information with kernel thread level profile information based on whether the identified kernel thread has been reused, the Final Office Action maintains that Figure 22, element 2216 teaches this feature and cites column 24, lines 15-30, which read as follows:

> The device driver then adds the computed difference to the thread-relative elapsed time for the current thread (step 2214) and updates the previous saved time for the current thread by storing the current time as the previous saved time, thereby maintaining a timestamp of the timepoint at which the thread-relative elapsed time was last updated (step 2216).
>
> The device driver then returns the thread-relative elapsed time (step 2218), and the profiler receives the thread-relative elapsed time (step 2220). The profiler then uses the thread-relative elapsed time for various profiling purposes, such as outputting a value in an event trace record in the trace buffer for recording periods of time between events (step 2222). A post-processor may then use the recorded times in the trace file to attribute various execution times to modules or routines. The process is then complete with respect to obtaining thread-relative elapsed time from the device driver.

This section of Berry merely teaches the updating of a thread-relative times within the Berry system. Appellants do not deny that Berry teaches updating profile information for kernels. However, nowhere in Berry is there any teaching regarding updating profile information with kernel thread level profile information based on whether an identified kernel thread has been reused. Neither Figure 22, the above cited section, nor any other section of Berry teaches to update profile information based on whether an identified kernel thread has been reused.

The Examiner has merely found general concepts of thread-relative metrics and updating of metrics in Berry and used these general concepts to allege that the specific features of claim 1 have been anticipated. Appellants respectfully submit that the Examiner has not fully considered the specific features set forth in the claims. If the Examiner had fully considered these features, there is no possibility that the Examiner would conclude that the general concepts of thread-relative metrics and updating of metrics anticipates a feature of determining whether an identified kernel thread has been reused and updating profile information with kernel thread level profile information based on whether the identified kernel thread has been reused. To the contrary, rather than finding these specific features, the Examiner has merely found "notification of event occurrences" and "adding, updating and maintaining information regarding threads and also time stamp for the thread which it places in a trace record" and has alleged that these general concepts somehow anticipate the specific features of claim 1. This is clearly not anticipation under 35 U.S.C. § 102(e) nowhere in Berry is there even any mention of kernel thread reuse, the recognition of kernel thread reuse being a problem with accurate profiling, or manner by which kernel thread reuse may be determined.

Thus, in view of the above, Appellants respectfully submit that Berry does not teach each and every feature of independent claim 1 as is required under 35 U.S.C. § 102(e). At least by virtue of their dependency on claim 1, Berry does not teach each and every feature of dependent claims 2-11. In addition to their dependency on claim 1, Berry does not teach each and every specific feature recited in dependent claims 2-9.

For example, with regard to claim 2, Berry does not teach checking a hash table for an entry of the identified kernel thread. The Final Office Action alleges that this feature is taught at column 10, lines 55-57 which read as follows:

> In post-processing phase 504 B-trees and/or hash tables may be employed to maintain names associated with records in the trace file to be processed.

While this section of Berry mentions that a hash table may be used to maintain names associated with records of a trace file, there is no teaching in this section of checking a hash table for an entry of an identified kernel thread. The general teaching of using hash tables to store names associated with records in a trace file does not teach or even suggest to one of ordinary skill in the art to check a hash table for an entry corresponding to an identified kernel thread. Moreover, merely teaching a hash table does not teach or even suggest to check a hash table for an entry corresponding to an identified kernel thread as part of determining whether an identified kernel thread has been reused.

The Final Office Action fails to even address these arguments in the Examiner's response to Appellants' arguments. Thus, the Examiner has failed to establish a prima facie case of anticipation with regard to claim 2.

Regarding claim 3, Berry does not teach applying kernel thread level profile information for a reused identified kernel thread to one of a previous application thread and a new application thread. As previously discussed above, Berry does not teach determining whether a kernel thread is reused and thus, cannot teach the features of claim 3. Moreover, the Final Office Action alleges that the features of claim 3 are taught by Berry in Figure 22 as element 2216. Element 2216 of the flowchart in Figure 22, which has been addressed above with regard to claim 1, reads "Device Driver Updates Previously Saved Time for Current Thread By storing Current Time as Previously Saved Time." This function associated with element 2216 has nothing to do with whether a kernel thread is reused or not and thus, cannot teach the features of claim 3 since these features reference "a reused identified kernel thread," let alone a "previous application thread" or "a new application thread."

As discussed above, Berry is completely concerned with application thread switches. Berry has no concern for kernel thread reuse. Thus, Berry has no reason to apply kernel thread level profile information for a reused kernel thread to one of a previous application thread and a new application thread. To the contrary, Berry is only concerned with determining timing metrics for the application threads.

Again, the Examiner fails to rebut Appellants arguments with regard to claim 3 in the Final Office Action. Thus, Appellants respectfully submit that the Examiner has not set forth a prima facie case of anticipation with regard to the features of claim 3.

With regard to claim 4, Berry does not teach applying kernel thread level profile information for a reused identified kernel thread to one of a previous application thread and a new application thread, marking the previous application thread being transmitted or associating the reused identified kernel thread with the new application thread in the hash table. As mentioned above with regard to claim 3, Berry does not teach identifying a kernel thread as being reused and thus, cannot teach the features of applying the kernel thread level profile information for the reused identified kernel thread to one of a previous application thread and a new application thread.

In addition, Berry does not teach marking a previous application thread as being transmitted. The Final Office Action alleges that this feature is taught in Figure 23 as element 2302 which reads "Device Driver Notified of Thread Switch from Previous Thread to Current Thread." This element does not teach marking a previous application thread as having been transmitted. To the contrary, all this element teaches is that the functionality of the flowchart illustrated in Figure 23 is initiated when the device driver is informed that a thread switch has occurred.

Moreover, Berry does not teach associating a reused identified kernel thread with a new application thread in a hash table. The Office Action alleges that this feature is taught in element 2306 of Figure 23 which reads "Device Driver Computes Difference Between Current Time and Previous Event Time for Previous Thread." This has nothing to do with associating a reused kernel thread with a new application thread in a hash table. To the contrary, all this element teaches is the calculation of a difference in event times.

As with claims 2 and 3 above, the Examiner fails to rebut Appellants arguments with regard to claim 4 in the Final Office Action. Thus, Appellants respectfully submit that the Examiner has not set forth a prima facie case of anticipation with regard to the features of claim 4.

The distinctions of the specific features of claims 2-4 are exemplary of the deficiencies of the Berry reference. Dependent claims 5-11 recite additional features that are likewise, not taught by the Berry reference. For example, claim 5 recites that the profile information is stored in a node for an application thread which is associated with the kernel thread level profile information. Berry does not even teach kernel thread level profile information but instead, is only concerned with application threads, i.e. threads that are executed by a processor as part of an application. Nowhere in Berry is there any teaching of a node for an application thread that is associated with kernel thread level profile information.

The Final Office Action alleges that this feature is taught by Berry at column 23, lines 57-60 which read as follows:

> The device driver may keep thread-relative information, such as thread-relative metrics, in a variety of data structures, such as trees, linked lists, hash tables, etc. As thread is dispatched, a node may be created for a newly dispatched thread, and when the thread terminates, the node may be either deleted or kept in memory for diagnostic profiling purposes.

While this section of Berry teaches the concept of nodes for application threads, there is no teaching or suggestion in this section, or any other section of Berry, regarding profile information being stored in a node for an application thread which is associated with the kernel thread level profile information. As with the other features of the claims above, the Examiner has merely found a general concept of a node for an application thread and attempted to reject the specific feature of claim 5 based on a generality rather than completely considering the features of the claim. Appellants have not merely claimed a node for an application thread in claim 5. To the contrary, claim 5 recites storing profile information in a node for an application thread which is associated with the kernel thread level profile information. This feature is not taught or even suggested by Berry.

Claim 6 recites sending a request for kernel thread level profile information to an operating system kernel, receiving the request for the kernel thread level profile information, accessing a processor data area containing processor level accumulated profile information, calculating a change in processor level accumulated profile information, accessing kernel thread level profile information held by the operating system kernel, updating kernel thread level profile information held by the operating system kernel with the change in processor level accumulated profile information, and sending the kernel thread level profile information held by the operating system kernel to a requestor. The Final Office Action alleges that these features are taught in column 3, lines 35-40, column 10, lines 35-55, column 15, lines 20-40, 55-60, column 16, lines 25-30, and column 23, lines 53-55. These sections of Berry read as follows:

> A method and system for maintaining a thread-relative metric for trace data using device driver support is provided. A profiling process may detect a current event and in response to the current event, may request an elapsed metric

since a preceding event. The profiling process then receives a thread-relative elapsed metric and may output a trace record for the current event in which is stored a metric equal to the received thread-relative elapsed metric.
(Column 3, lines 35-40)

...may originate form a stack walking function executed in response to a timer interrupt, e.g., a stack unwind record, also called a call stack record.

For example, the following process may occur during the profiling phase if the user of the profiling utility has requested sample-based profiling information. Each time a particular type of timer interrupt occurs, a trace record is written, which indicates the system program counter. This system program counter may be used to identify the routine that is interrupted. In the depicted example, a timer interrupt is used to initiate gathering of trace data. Of course, other types of interrupts may be used other than timer interrupts. Interrupts based on a programmed performance monitor event or other types of periodic events may be employed. In the post-processing phase 504, the data collected in the buffer is sent to a file for post-processing. IN one configuration, the file may be sent to a server, which determines the profile for the processes on the client machine. Of course, depending on available resources, the post-processing also may be performed on the client machine.
(Column 10, lines 35-55)

The post-processing of a trace file may result in a report consisting of three kinds of time spent in a routine, such as routine A: (1) base time – the time spent executing code in routine A itself; (2) cumulative time (or "cum time" for short) – the time spent executing in routine A plus all the time spent executing every routine that routine A calls (and all routines they call, etc.); and (3) wall-clock time or elapsed time. This type of timing information may be obtained from event-based trace records as these records have timestamp information for each record.

A routine's cum time is the sum of all the time spent executing the routine plus the time spent executing any other routines while that routine is below it on the call stack. In the example above in FIG. 10C, routine A's base time is 2 ms, and its cum time is 10 ms. Routine B's base time is 8 ms, and its cum time is also 8 ms because it does not call any other routines. It should be noted that cum time may not be generated if a call stack tree is being generated on-the-fly – cum time may only be computed after the fact during the post-processing phase of the profile utility.

For wall-clock or elapsed time, if while routine B was running, the system fielded an interrupt or suspended this thread to run another thread, or if routine B blocked waiting on a lock or I/O...
(Column 15, lines 20-40)

Although base time, cum time and elapsed time were defined in terms of processor time spent in routines, sample based profiling is useful for attributing consumption of almost any system resource to a set of routines, as described in

more detail below with respect to FIG. 11B. Referring to FIG. 10C again, if routine A initiated two disk I/O's, and then routine B initiated three more I/O's when called by routine A, routine A's "base I/O's" are two and routine A's "cum I/O's" are five.
(Column 15, lines 55-60)

As noted above, this type of timing information may be obtained from event-based trace records as these records have timestamp information for each record. The address represents a function entry point. The base time represents the amount of time consumed directly by this thread executing this function. The cumulative time is the amount of time consumed by this thread execution this function and all functions below it on the call stack.
(Column 16, lines 25-30)

Profiler 2104 may contain the functionality necessary for generating trace records that are output to a trace buffer. Profiler 2104 may request and receive certain types of system information from device driver or kernel 2106.
(Column 23, lines 53-55)

These sections of Berry merely generally teach profiling of an application execution with various metrics, e.g., cum time, base time, etc., and the storing of these metrics in trace records that may be output to a trace buffer. These sections also state that the profiler may obtain information from the operating system kernel. What is lacking in these sections of Berry is any teaching that has any relevance to the specific features of claim 6. That is, there is nothing in these sections, or any other section, of Berry that teaches sending a request for kernel thread level profile information to an operating system kernel, updating kernel thread level profile information held by the operating system kernel with a change in processor level accumulated profile information, and sending the kernel thread level profile information held by the operating system kernel to a requestor. Nowhere in Berry is there any teaching to update kernel thread level profile information with a change in processor level accumulated profile information.

Regarding claim 7, Berry does not teach resetting kernel thread level profile information held by an operating system kernel. The Final Office Action alleges that this feature is taught at column 25, lines 1-4 of Berry which read as follows:

...device driver computes the thread-relative elapsed time for thread A and saves it for subsequent processing. The device driver also sets the previous event time for thread A to the current time, i.e. the time at which the thread switch occurs.

All that this section of Berry teaches is setting the previous event time to the current time. There is no mention here, or anywhere else in Berry, that kernel thread level profiling information held by an operating system kernel is reset. Once again, the Examiner merely found the general concept of setting a value and alleges that this anticipates the specific features of claim 7. Claim 7 is clearly not anticipated by a general teaching of setting a previous event time to a current time.

With regard to claim 8, Berry does not teach updating an information area for a current application thread based on the identified kernel thread having not been reused. As discussed at length above, Berry does not teach determining whether a kernel thread has been reused or not and thus, cannot teach updating an information area for a current application thread based on the identified kernel thread having not been reused. However, the Final Office Action alleges that this feature is taught by Berry at column 17, lines 3-5 which have been discussed above.

As discussed above, this section of Berry merely discusses a recursion depth. Recursion depth is not the same as thread reuse as discussed above. Moreover, merely teaching recursion depth does not in itself teach updating an information area for a current application thread based on the identified kernel thread having not been reused. How can determining a recursion depth teach to update a current application thread based on a kernel thread having not been reused? The two concepts are completely different and have nothing to do with one another. Again, the Examiner has failed to consider the specific features recited in the claim and has not established a prima facie case of anticipation with regard to claim 8.

As with claim 8, Berry does not teach the specific features of claim 9 which recite updating an information area for a previous application thread based on the identified kernel thread having been reused. This is the opposite feature from that of claim 8 but similar reasoning applies. That is, simply by teaching recursion depth in column 17, lines 3-5, Berry has not taught the features of claim 9. There is no correlation between recursion depth and kernel thread reuse. Furthermore, merely determining recursion depth does not teach updating an information area for a previous application thread based on the identified kernel thread.

Thus, all of claims 2-9 recite additional features that are not taught by Berry and are therefore allowable over Berry by virtue of the specific features recited in these claims.

B. Claims 12, 14-18, 21, 23-28

With regard to independent claims 12, 21 and 28, the Final Office Action merely states to refer to the rejection of claim 1 above. Appellants first wish to point out that claims 12, 21 and 28 are of a different scope than claim 1. For example, claim 12, which is representative of the other independent claims 21 and 28, reads as follows:

> 12. A method for monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:
> receiving a value of a metric variable for a kernel thread;
> determining if the kernel thread has been previously used by a first application thread; and
> applying the value of the metric variable to a second application thread if the kernel thread has been previously used by the first application thread.
> (emphasis added)

Berry does not teach determining if a kernel thread has been previously used by a first application thread. In fact, Berry does not teach kernel threads being used by application threads and thus, cannot teach determining if a kernel thread has been previously used by a first application thread. To the contrary, the system of Berry is directed to identifying switches in application threads and resolving event timing based on the detection of an application thread switch. There is no teaching in Berry regarding kernel threads or determining if a kernel thread has been previously used by a first application thread. Since Berry does not teach determining if the kernel thread has been previously used by a first application thread, there cannot be any teaching in Berry regarding applying a value of a metric variable for a kernel thread to a second application thread when it is determined that the kernel thread has been previously used by a first application thread.

The only sections referenced by the Final Office Action with regard to claims 12, 21 and 28 are the same sections discussed above with regard to claim 1 (since the Office Action fails to explicitly address the features of claims 12, 21 and 28 and instead rests on the rejection of claim 1). These sections have been shown above to only teach maintaining information regarding how many times a routine calls itself (recursive entry) and determining a time metric during an application thread switch. Recursive entry of a routine has nothing to do with determining whether a kernel thread has been previously used by a first application thread and then applying the value of a metric variable to a second application thread if the kernel thread has been previously used by a first application thread. Similarly, application thread switches are only directed to switching between processing of one application thread to another application thread and have nothing to do with determining whether a kernel thread has been previously used by a first application thread.

In view of the above, Appellants respectfully submit that Berry does not teach each and every feature of independent claims 12, 21 and 28. At least by virtue of their dependency on claims 12 and 21, respectively, Appellants respectfully submit that Berry does not teach each and every feature of dependent claims 14-18 and 23-27.

In addition to the above, Berry does not teach the specific features recited in dependent claims 14, 16-18, 23 and 25-27. For example, with regard to claims 14 and 23, Berry does not teach identifying the first application thread as being terminated based on the kernel thread having been previously used by the first application thread. The Final Office Action alleges that the features of these claims are taught by Berry in Figure 23, elements 2302 and 2306, the text of which is reproduced above (see rejection of claim 4 discussed above). Neither of these steps in the flowchart of Figure 23 have anything to do with identifying an application thread as being terminated based on a determination that a kernel thread has been previously used by the application thread. To the contrary, these steps recite the functions within Berry of the device driver being notified of a thread switch from a previous thread to a current thread and the device driver computing the difference between a current time and a previous even time for a previous thread. There is nothing in these steps that teaches to determine that an application thread has been terminated based on a kernel thread having been previously used by the application thread.

With regard to claims 16 and 25, Berry does not teach comparing an identity of a kernel thread to a list of identities of previously used kernel threads. The Final Office Action again alleges that this feature is taught in element 2306 of Figure 3 in Berry. As discussed above, all this element teaches is that a device driver computes the difference in a current time and a previous event time for a previous thread. It is not at all clear how this has anything to do with comparing the identity of a kernel thread to a list of identities of previously used kernel threads. In fact, nowhere in Berry is there ever a comparison of the identity of a kernel thread to a list of identifies of previously used kernel threads. There is no teaching of such a feature because Berry is not concerned with reuse of kernel threads and thus, there is no reason to perform such a comparison.

As with each of the other rejections, other than the rejection of claim 1, the Examiner fails to provide any rebuttal of Appellants' arguments that the Berry reference does not teach the specific features recited in claims 14, 16, 23 and 25. Thus, the Examiner has failed to establish a prima facie case of anticipation with regard to claims 14, 16, 23 and 25, in addition to a similar failure to establish a prima facie case in each of the other rejections set forth in the Final Office Action.

The distinctions of the specific features of claims 14, 16, 23 and 25 are exemplary of the deficiencies of the Berry reference. Dependent claims 15, 17-18, 24 and 26-27 recite additional features that are likewise, not taught by the Berry reference. Thus, all of claims 14-18 and 23-27 recite features that are not taught by Berry and are therefore allowable over Berry.

C. Claims 19-20

With regard to independent claim 19, the Final Office Action merely states to refer to the rejection of claim 1 above. Appellants first wish to point out that claim 19 is of a different scope than claim 1. Claim 19 reads as follows:

> 19. A method for monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:
> receiving a plurality of values of metric variables for a plurality of kernel threads;
> for each kernel thread:
> determining if a kernel thread has been previously used by a prior application thread; and
> applying the value of the metric variable for the kernel thread to a current application thread if the kernel thread has been previously used by a prior application thread. (emphasis added)

As previously noted above, Berry does not teach determining if a kernel thread has been previously used by an application thread. Berry also does not teach applying a value of a metric variable for a kernel thread to a current application thread if the kernel thread has been previously used by a prior application thread. The Examiner is referred to the above discussed of independent claims 12, 21 and 28, which recite similar, although not the same, features. The teaching of maintaining information regarding recursive entries into a routine has nothing to do with the features emphasized above in claim 19, as previously discussed. Furthermore, merely teaching the determination of timing metrics during application thread switches also does not anticipate the particular emphasized features above in claim 19 for similar reasons as stated previously.

Thus, Berry does not teach the features of claim 19. Furthermore, the Examiner has failed to rebut the arguments with regard to claim 19 and thus, has failed to establish a prima facie case of anticipation with regard to claim 19.

Furthermore, with regard to dependent claim 20, Berry does not teach that a plurality of values of metric variables for a plurality of kernel threads are received from an operating system kernel wherein each of the plurality of values was contained in a linked list of entries, each entry in the linked list being a value of a metric variable for a specific kernel thread. The Final Office Action alleges that this feature is taught by Berry simply because Berry, in column 23, lines 55-60 states that the device driver may keep thread-relative information in linked lists. While Berry teaches linked lists as a means for keeping thread-relative information, the thread relative information referred to by Berry is application thread-relative information. The application thread-relative information of Berry does not include metric variables for a plurality of kernel threads and this information is not received from an operating system kernel. Furthermore, even though a linked list may be taught by Berry, there is no teaching in Berry that each entry in a linked list is a value of a metric variable for a specific kernel thread.

Therefore, Appellants respectfully submit that Berry does not teach each and every feature of claims 19-20 as is required under 35 U.S.C. § 102(e).

D. Claims 29-34

Claim 29 recites that a value of a metric variable for the kernel thread is received in response to an event occurring based on the execution of a second application thread. Claim 30 recites that applying the value of the metric variable to a second application thread includes increasing a current metric value for the second application thread in a node associated with the second application thread by an amount equal to the value of the metric variable. The Final

Office Action rejects claims 29 and 30 because Berry allegedly teaches a "current thread" in column 3, lines 35-50, which has been discussed above. Simply reciting a "current thread" does not render the specific features of claims 29 and 30 anticipated. Furthermore, simply teaching generating a thread-relative elapsed metric by obtaining a stored metric for a preceding event of a current thread, a current metric and computing the difference in response to notification of an event, as generally taught in column 3, lines 35-50, does not anticipate a value of a metric variable for the kernel thread being received in response to an event occurring based on the execution of a second application thread or applying the value of the metric variable to a second application thread including increasing a current metric value for the second application thread in a node associated with the second application thread by an amount equal to the value of the metric variable.

While Berry teaches the determination of timing metrics during thread switches, Berry does not teach that these timing metrics are values of a metric variable for a kernel thread or that a value for a metric variable for a kernel thread is received in response to an event occurring based on the execution of a second application thread. Again, Berry is only concerned with metrics for application threads and is not concerned at all with kernel threads, let alone receiving a value for a metric variable for a kernel thread in response to execution of a second application thread, as recited in claim 29.

Furthermore, Berry does not teach increasing a current metric value for a second application thread in a node associated with the second application thread by an amount equal to the value of the metric variable for a kernel thread. To the contrary, as discussed above, Berry is only concerned with application thread switches. Thus, there is no consideration of kernel threads in Berry, let alone increasing metric values for an application thread by an amount equal to the value of a metric variable for a kernel thread, as recited in claim 30. Nothing in Berry teaches the features of claims 29 and 30.

With regard to claims 31-34, these claims are similar to claims 29 and 30, respectively, but are dependent from other independent claims. Thus, similar distinctions as those discussed with regard to claims 29 and 30 apply to claims 31-34. Furthermore, since all of claims 29-34 are dependent upon respective independent claims, these claims are also allowable over Berry by virtue of their dependency.

CONCLUSION

In view of the above, Appellants respectfully submit that all of claims 1-12, 14-21 and 23-34 are allowable over Berry and that the application is in condition for allowance. Accordingly, Appellants respectfully request the Board of Patent Appeals and Interferences to overturn the rejections set forth in the Final Office Action.

Respectfully submitted,

[illegible]

Stephen J. Walder, Jr.
Reg. No. 41,534
Yee & Associates, P.C.
PO Box 802333
Dallas, TX 75380
(972) 367-2001

APPENDIX OF CLAIMS

The text of the claims involved in the appeal reads:

1. A method for monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:

receiving an event indication;

ascertaining kernel thread level profile information;

identifying a kernel thread, wherein the kernel thread level profile information is attributed to the identified kernel thread;

determining whether the identified kernel thread has been reused; and

updating profile information with the kernel thread level profile information based whether the identified kernel thread has been reused.

2. The method recited in claim 1 above, wherein determining whether the identified kernel thread has been reused further comprises:

checking a hash table for an entry of the identified kernel thread.

3. The method recited in claim 1 above, wherein updating profile information with the kernel thread level profile information based on the identified kernel thread is reused further comprises:

applying the kernel thread level profile information for the reused identified kernel thread to one of a previous application thread and a new application thread.

4. The method recited in claim 2 above, wherein updating profile information with the kernel thread level profile information based on the identified kernel thread is reused further comprises:

applying the kernel thread level profile information for the reused identified kernel thread to one of a previous application thread and a new application thread;

marking the previous application thread being terminated; and

associating the reused identified kernel thread with the new application thread in the hash table.

5. The method recited in claim 1 above, wherein the profile information is stored in a node for a application thread which is associated with the kernel thread level profile information.

6. The method recited in claim 1 above, wherein ascertaining kernel thread level profile information further comprises:

sending a request for kernel thread level profile information to an operating system kernel, wherein the operating system kernel responds to the request by:

receiving the request for kernel thread level profile information;

accessing a processor data area containing processor level accumulated profile information;

calculating a change in processor level accumulated profile information;

accessing kernel thread level profile information held by the operating system kernel;

updating kernel thread level profile information held by the operating system kernel with the change in processor level accumulated profile information; and sending the kernel thread level profile information held by the operating system kernel to a requestor.

7. The method recited in claim 5 above further comprises:

resetting the kernel thread level profile information held by the operating system kernel.

8. The method recited in claim 1 above, wherein updating profile information with the kernel thread level profile information based whether the identified kernel thread has been reused further comprises:

updating an information area for a current application thread based on the identified kernel thread having not been reused.

9. The method recited in claim 1 above, wherein updating profile information with the kernel thread level profile information based whether the identified kernel thread has been reused further comprises:

updating an information area for a previous application thread based on the identified kernel thread having been reused.

10. The method recited in claim 1 above, wherein updating profile information with the kernel thread level profile information based whether the identified kernel thread has been reused further comprises:

accessing an information area for a application thread based on the identified kernel thread;

updating an information area for the application thread.

11. The method recited in claim 3 above, wherein the application is a Java application.

12. A method for monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:

receiving a value of a metric variable for a kernel thread;

determining if the kernel thread has been previously used by a first application thread; and

applying the value of the metric variable to a second application thread if the kernel thread has been previously used by the first application thread.

14. The method recited in claim 12 above, further comprises:

identifying the first application thread as being terminated based on the kernel thread having been previously used by the first application thread.

15. The method recited in claim 12 above, wherein the first application thread is a current application thread based on the kernel thread having not been used.

16. The method recited in claim 12 above, wherein determining if the kernel thread has been previously used by the first application thread further comprises:

comparing an identity of the kernel thread to a list of identities of previously used kernel threads.

17. The method recited in claim 12 above, wherein the value of a metric variable for a kernel thread is a change in value of the metric variable since the last receipt of the metric variable for the kernel thread.

18. The method recited in claim 12 above, wherein the metric variable relates to one of allocation bytes, allocation objects, time, live object and live bytes.

19. A method for monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:

receiving a plurality of values of a metric variables for a plurality of kernel threads;

for each kernel thread:

determining if a kernel thread has been previously used by a prior application thread; and

applying the value of the metric variable for the kernel thread to a current application thread if the kernel thread has been previously used by a prior application thread.

20. The method recited in claim 19 above, wherein the plurality of values of metric variables for a plurality of kernel threads are received from an operating system kernel wherein each of the plurality of values was contained in a linked list of entries, each entry in the linked list being a value of a metric variable for a specific kernel thread.

21. A data processing system for monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:

receiving means for receiving a value of a metric variable for a kernel thread;

determining means for determining if the kernel thread has been previously used by a first application thread; and

applying means for applying the value of the metric variable to a second application thread if the kernel thread has been previously used by the first application thread.

23. The system recited in claim 22 above, further comprises:

identifying means for identifying the first application thread as being terminated based on the kernel thread having been previously used by the first application thread.

24. The system recited in claim 21 above, wherein the first application thread is a current application thread based on the kernel thread having not been used.

25. The system recited in claim 21 above, wherein the determining means for determining if the kernel thread has been previously used by the first application thread further comprises:

comparing means for comparing an identity of the kernel thread to a list of identities of previously used kernel threads.

26. The system recited in claim 21 above, wherein the value of a metric variable for a kernel thread is a change in value of the metric variable since the last receipt of the metric variable for the kernel thread.

27. The system recited in claim 21 above, wherein the metric variable relates to one of allocation bytes, allocation objects, time, live object and live bytes.

28. A computer program product for implementing a method for monitoring performance of a program being executed using per thread metric variables with reused kernel threads comprising:

receiving instructions for receiving a value of a metric variable for a kernel thread;

determining instructions for determining if the kernel thread has been previously used by a first application thread; and

applying instructions for applying the value of the metric variable to a second application thread if the kernel thread has been previously used by the first application thread.

29. The method of claim 12, wherein the value of the metric variable for the kernel thread is received in response to an event occurring based on the execution of the second application thread.

30. The method of claim 12, wherein applying the value of the metric variable to the second application thread includes increasing a current metric value for the second application thread in a node associated with the second application thread by an amount equal to the value of the metric variable.

31. The data processing system of claim 21, wherein the value of the metric variable for the kernel thread is received in response to an event occurring based on the execution of the second application thread.

32. The data processing system of claim 21, wherein the applying means for applying the value of the metric variable to the second application thread includes means for increasing a current metric value for the second application thread in a node associated with the second application thread by an amount equal to the value of the metric variable.

33. The computer program product of claim 28, wherein the value of the metric variable for the kernel thread is received in response to an event occurring based on the execution of the second application thread.

34. The computer program product of claim 28, wherein the applying instructions for applying the value of the metric variable to the second application thread include instructions for increasing a current metric value for the second application thread in a node associated with the second application thread by an amount equal to the value of the metric variable.